CS 6110 Lecture 27 Soundness of the Simply-Typed Lambda Calculus 1 April 2013

Lecturer: Andrew Myers

1 Soundness from the operational perspective

We have seen that we can write useful typing rules, but how do we know we got them right? This depends on what the type system is supposed to accomplish. The traditional application is making sure that programs don't have run-time type errors (or, at least, helping the programmer avoid them). A type system is *sound* if well-typed programs do not incur run-time type errors; that is, they do not get stuck when evaluated according to the operational semantics:

$$\textit{Typing rules are sound} \iff \textit{no well-formed program gets stuck.}$$

Note that we use well-formed and well-typed equivalently for this language. To be more precise,

$$\vdash e : \tau \wedge\ e \longrightarrow^* e' \implies e' \in \textbf{Value} \vee \exists e''. e' \longrightarrow e''$$

Even with the fanciest type systems, we can't avoid difficult reasoning about some kinds of errors. For example, in OCaml, we can still get nonexhaustive matches and numeric overflows.

2 Soundness of the simply-typed lambda calculus

We will show soundness of $\lambda^{\rightarrow}$ in two steps, using the following two lemmas:

- *Preservation*: As we evaluate a program, its type is preserved at each step. This property is sometimes also called *subject reduction*, for historical reasons.

$$\vdash e : \tau\ \wedge\ e \longrightarrow e' \implies\ \vdash e' : \tau$$

- *Progress*: Every program is either a value or can be stepped into another program (and using *Preservation*, this will be of the same type):

$$\vdash e : \tau \implies e = v \in \textbf{Value} \vee \exists e''. e \longrightarrow e''$$

Note that with these two lemmas soundness easily follows. Preservation says every step preserves the type, so we use induction on the number of steps taken in $e \longrightarrow^* e'$ to show that e' must have the same type as e. Then Progress can be applied to e' to show that evaluation isn't stuck there. We will now set out to prove these two lemmas.

2.1 Proof of Preservation

Assuming $e : \tau \wedge e \longrightarrow e'$ we need to show that $\vdash e' : \tau$. We do this by well-founded induction on typing derivations. (Note that the typing derivations are finite and therefore the relation of subderivation is well-founded.) The property we are trying to show on typing derivations is:

$$P(\vdash e : \tau) \iff \forall e'. e \longrightarrow e' \implies \vdash e' : \tau$$

Given that we know $e \longrightarrow e'$, there are three possibilities corresponding to the three evaluation rules:

$$\frac{e_0 \longrightarrow e_0'}{e_0\ e_1 \longrightarrow e_0'\ e_1}\ (L) \qquad \frac{e_1 \longrightarrow e_1'}{v_0\ e_1 \longrightarrow v_0\ e_1'}\ (R) \qquad \frac{}{(\lambda x\!:\!\tau.\, e)\ v \longrightarrow e\{v/x\}}\ (\beta)$$

- Case (L): $e = e_0\ e_1$.

  Because we have a typing derivation for $e_0\ e_1$, we know that there are typing derivations for e_0 too. We must have $\vdash e_0 : \tau_1 \rightarrow \tau$ and $\vdash e_1 : \tau_1$ for some type τ_1. Because the typing derivation for e_0 is a subderivation of the typing derivation, by the induction hypothesis the step $e_0 \longrightarrow e_0'$ also preserves type. So we know $\vdash e_0' : \tau_1 \rightarrow \tau$, and we now have both of the premises necessary to construct the proof that $e_0'\ e_1$ has the desired type τ.

- Case (R): $e = v_0\ e_1$

  This case is symmetrical to case (L); in this case e_1 is the subexpression making the step.

- Case 3: $e = (\lambda x\!:\!\tau_1.\, e_0')\ v_1$

  The typing derivation of $\vdash e : \tau$ must look like this:

$$\dfrac{\dfrac{x\!:\!\tau_1 \vdash e_0' : \tau}{\vdash (\lambda x\!:\!\tau_1.\, e_0') : \tau_1 \to \tau} \quad \vdash v_1 : \tau_1}{\vdash (\lambda x\!:\!\tau_1.\, e_0')\ v_1 : \tau}$$

  We know that $e \longrightarrow e_0'\{v_1/x\}$, so we need to show that $e_0\{v_1/x\} : \tau$ using the facts $x : \tau_1 \vdash e_0' : \tau$ and $\vdash v_1 : \tau_1$. Our induction hypothesis doesn't help us here; we need to prove this separately. It follows as a special case of a substitution lemma that captures the type preservation of β-reduction:

2.2 Substitution lemma

$$\Gamma, x\!:\!\tau' \vdash e : \tau \ \wedge\ \vdash v : \tau' \implies \Gamma \vdash e\{v/x\} : \tau$$

We prove this by induction on the typing derivation of e.

- Case 1: $e = b$. In this case the result is trivial.
- Case 2: a variable
  - Case 2a: $e = x$. Then $e\{v/x\} = v$ and $\tau' = \tau$. We have $\vdash v : \tau$ and need $\Gamma \vdash v : \tau$. This holds because adding new things to the typing context doesn't cause us to lose the ability to produce a typing derivation. This property can be expressed in the following *weakening lemma* whose proof we leave as an exercise:

$$\Gamma \vdash e : \tau \ \wedge\ x \notin FV(e) \implies \Gamma, x\!:\!\tau' \vdash e : \tau$$

    Then we can get from $\emptyset \vdash v : \tau$ to $\Gamma \vdash v : \tau$ by applying the weakening lemma enough times to add in the finite number of bindings in Γ.
  - Case 2b: $e = y \neq x$. In this case $e\{v/x\} = e$ and therefore the result trivially holds.
- Case 3: $e = e_0\ e_1$. Using the definition of substitution, $e\{v/x\} = e_0\{v/x\}\ e_1\{v/x\}$.

  Using the derivation of $\Gamma, x\!:\!\tau' \vdash e : \tau$ we have $\Gamma, x\!:\!\tau' \vdash e_0 \!:\! \tau_1 \to \tau$ and $\Gamma, x\!:\!\tau' \vdash e_1 \!:\! \tau_1$:

$$\dfrac{\Gamma, x\!:\!\tau' \vdash e_0 \!:\! \tau_1 \to \tau \quad \Gamma, x\!:\!\tau' \vdash e_1 \!:\! \tau_1}{\Gamma, x\!:\!\tau' \vdash e \!:\! \tau}$$

  Therefore we can use the induction hypothesis to obtain $\Gamma \vdash e_0\{v/x\} : \tau_1 \to \tau$ and $\Gamma \vdash e_1\{v/x\} : \tau_1$. This therefore implies $\Gamma \vdash e\{v/x\} \!:\! \tau$ via the application rule.
- Case 4: a lambda abstraction
  - Case 4a: $e = \lambda x\!:\!\tau''.\, e_2$.

    Then $e\{v/x\} = e$. From the typing derivation of form $\Gamma, x\!:\!\tau' \vdash \lambda x\!:\!\tau''.\, e_2 : \tau$ where $\tau = \tau'' \to \tau_2$, and the fact $\Gamma, x\!:\!\tau', x\!:\!\tau''$ = $\Gamma, x\!:\!\tau''$, we have $\Gamma, x\!:\!\tau'' \vdash e_2 : \tau_2$, from which we can derive $\Gamma \vdash \lambda x\!:\!\tau''.\, e_2 : \tau'' \to \tau_2$, as desired.
  - Case 4b: $e = \lambda y\!:\!\tau''.\, e_2$.

    Note that $y \notin FV(v)$, so $e\{v/x\} = \lambda y : \tau''.\, e_2\{v/x\}$ and the typing of e looks like the following, where $\tau = \tau'' \to \tau_2$:

$$\dfrac{\Gamma, x\!:\!\tau', y\!:\!\tau'' \vdash e_2 \!:\! \tau_2}{\Gamma, x\!:\!\tau', \vdash (\lambda y\!:\!\tau''.\, e_2) \!:\! \tau'' \to \tau_2}$$

    Therefore, we can derive $\Gamma, y : \tau'', x : \tau' \vdash e_2 : \tau_2$ and use the inductive hypothesis to obtain $\Gamma, y : \tau'' \vdash e_2\{v/x\} : \tau_2$. Therefore, $\Gamma \vdash (\lambda y\!:\!\tau''.\, e_2\{v/x\}) \!:\! \tau$ and we are done.

3 Progress Lemma

To finish the proof of soundness for the typed lambda calculus, we weed to prove progress. The progress lemma captures the idea that we cannot get stuck when evaluating a well-formed expression.

Progress: $\vdash e : \tau \implies e \in$ **Value** $\vee\, \exists e'.e \longrightarrow e'$
Proof: We use induction on the typing derivation of e. Remember the definition of an expression in $\lambda^{\rightarrow}$:

$$e ::= b \mid x \mid \lambda x\!:\!\tau.\, e \mid e_0\, e_1$$

There are four cases:

- Case $e = b$: Trivial: $b \in$ **Value**.
- Case $e = x$: This case is not possible because we would have $\vdash x : \tau$ and from the empty environment we cannot assign any type to x.
- Case $e = \lambda x\!:\!\tau_0.\, e_1$: We have that $e \in$ **Value**.
- Case $e = e_0\, e_1$: We know that there is a typing derivation for $\vdash e_0\, e_1 : \tau$ and this derivation must have the form:

  $$\frac{\vdash e_0 : \tau' \quad \vdash e_1 : \tau' \to \tau}{\vdash e_0\, e_1 : \tau}$$

  By the induction hypothesis, $e_0 \in$ **Value** $\vee\, \exists e_0'.e_0 \longrightarrow e_0'$ and $e_1 \in$ **Value** $\vee\, \exists e_1'.e_1 \longrightarrow e_1'$.
  We have four possibilities now:

  - Both e_0 and e_1 are values. Since e_0 is a value with an arrow type, it has to be an abstraction. Say e_0 is $\lambda x \in \tau' \,.\, e_2$ and e_1 is some value v. Then

    $$e = (\lambda x \in \tau' \,.\, e_2)v \longrightarrow e_2\{v/x\}$$

    so, $e' = e_2\{v/x\}$ as desired.
  - e_0 is not a value. Then $\exists e_0'.e_0 \longrightarrow e_0'$ and we have

    $$\frac{e_0 \longrightarrow e_0'}{e_0\, e_1 \longrightarrow e_0' e_1}$$

  - e_0 is some value v, but e_1 is not a value. Then $\exists e_1'.e_1 \longrightarrow e_1'$ and we have

    $$\frac{e_1 \longrightarrow e_1'}{v\, e_1 \longrightarrow v\, e_1'}$$

This finishes the proof.

We have shown how to prove a very simple type system sound. However, this same approach works on the more complex type systems we will see shortly.